Deep Learning (IST, 2021-22)

Practical 4: Multilayer Perceptron and Backpropagation

André Martins, Andreas Wichert, Luis Sá-Couto

Question 1

Consider a network with four layers with the following numbers of units: $4, 4, 3, 3$. Assume all units, except the ones in the output layer, use the hyperbolic tangent activation function.

1. Initialize all connection weights and biases to 0.1. Using the squared error loss do a **stochastic gradient descent** update (with learning rate $\eta = 0.1$) for the training example:

$$\boldsymbol{x} = \begin{bmatrix} 1 & 0 & 1 & 0 \end{bmatrix}^{\mathsf{T}}, \quad \boldsymbol{y} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

2. Reusing the computations from the previous exercise do a **gradient descent** update (with learning rate $\eta = 0.1$) for the batch with the training example from the a) and the following:

$$\boldsymbol{x} = \begin{bmatrix} 0 & 0 & 10 & 0 \end{bmatrix}^{\mathsf{T}}, \quad \boldsymbol{y} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Question 2

Let us repeat the exact same exercise as in 1) but this time we will change:

- The output units have a softmax activation function
- The error function is cross-entropy

1. Initialize all connection weights and biases to 0.1. Using the cross-entropy loss do a **stochastic gradient descent** update (with learning rate $\eta = 0.1$) for the training example:

$$\boldsymbol{x} = \begin{bmatrix} 1 & 0 & 1 & 0 \end{bmatrix}^{\mathsf{T}}, \quad \boldsymbol{y} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

2. Reusing the computations from the previous exercise do a **gradient descent** update (with learning rate $\eta = 0.1$) for the batch with the training example from the a) and the following:

$$\boldsymbol{x} = \begin{bmatrix} 0 & 0 & 10 & 0 \end{bmatrix}^{\mathsf{T}}, \quad \boldsymbol{y} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Question 3

Now it's time to train a multi-layer perceptron on real data and see what happens.

1. Load the UCI handwritten digits dataset using `scikit-learn`:

   ```
   from sklearn.datasets import load_digits
   data = load_digits()
   ```

   This is a dataset containing 1797 8x8 input images of digits, each corresponding to one out of 10 output classes. You can print the dataset description and visualize some input examples with:

   ```
   print(data.DESCR)

   import matplotlib.pyplot as plt
   plt.gray()
   for i in range(10):
       plt.matshow(data.images[i])
   plt.show()
   ```

   Randomly split this data into training (80%) and test (20%) partitions. This can be done with:

   ```
   from sklearn.model_selection import train_test_split
   X_train, X_test, y_train, y_test = train_test_split(
       X, y, test_size=0.2, random_state=42)
   ```

2. Run your implementation of the multi-layer perceptron on this dataset for 100 epochs, using stochastic gradient descent with $\eta = 0.001$. Use a single hidden layer with 50 hidden units. Measure the training and test accuracy.

3. Use `scikit-learn`'s implementation of a single-layer multi-layer perceptron. This can be done with

   ```
   from sklearn.neural_network import MLPClassifier
   clf = MLPClassifier(hidden_layer_sizes=(50),
                       activation='tanh',
                       solver='sgd',
                       learning_rate='constant',
                       learning_rate_init=0.001,
                       nesterovs_momentum=False,
                       random_state=1,
                       max_iter=1000)
   clf.fit(X_train, y_train)
   print(clf.score(X_train, y_train))
   print(clf.score(X_test, y_test))
   ```

   Compare the resulting accuracies.